Politecnico Di Torino

ICT for Smart Societies

Lab Reports For:
Management and content delivery for Smart Networks algorithms and modelling

Authors:
Seyyed Sadegh Bibak Sareshkeh 235986
Giampaolo Marinaro 242195
Navid Yamini 235935

Professor:
Michela Meo
Idilio Drago

June 2017

Lab 1

Exercise 1: A Simplified Web Server

1-1- The M/M/1- Queue

In the first part, the web server is simulated with a M/M/1 queue. In this system, the packets' arrival rates are exponentially distributed with parameter λ and the service rate with parameter μ. While capacity of the queue is infinite and the service discipline is FIFO. The system will be ergodic if $\lambda < \mu$ or $\rho = (\lambda / \mu) < 1$.

We can calculate the mean number of the customers in the system by:

$$E[N] = \sum_{i=1}^{\infty} i\pi_i = \sum_{i=1}^{\infty} i(1-\rho)\rho^i = \rho(1-\rho)\sum_{i=1}^{\infty} i\rho^{i-1} =$$
$$= \rho(1-\rho)\frac{1}{(1-\rho)^2} = \frac{\rho}{1-\rho}$$

Equation 1: Mean number of the customer in M/M/1

And, the mean response time is given by:

$$E[T] = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda}$$

Equation 2: Mean response time foe M/M/1

According to above equations, the number of customers grows to infinity as ρ approaches to 1. On the other hand, if we increase the λ the response time grows to infinite.

1-1-1- Inputs

For starting the simulation, we set the input parameters as follows:

```
RANDOM_SEED = 40
INTER_ARRIVAL = 15.0
SERVICE_TIME = 0.1
NUM_SERVER = 1
SIM_TIME = 1000000
CONFIDENCE = 0.95
Landa = 1.0/INTER_ARRIVAL
mu = 1.0/SERVICE_TIME
```

1-1-2- Algorithm Description

The algorithm works in this way:

In every run, the time it takes for the packet to enter to the system and the time that it takes to departure is stored in a list and by calculating the differences between these two lists, the response time (the total time that a packet stays in a system) for every packet can be extracted. At the end, by averaging the values in the final list, the mean response time is calculated. For the buffer occupancy, the size of the queue is stored in a list every time that the state of the system changes (The state of the system changes when a packet enters or goes out from the queue). Like before, by calculating the average from the list we can find the average buffer occupancy for one run of simulation.

This procedure is repeated for 145 times and in every iteration, we increase the SERVICE_TIME by 0.1. Moreover, in every iteration the average response time and average buffer occupancy is saved in a vector. For the final step, after the last iteration, we have a mean response time vector and a set of average buffer occupancies. For plotting the results, we put these vectors as a y-axis and for x-axis, we use the value of ρ which is calculated in every iteration.

1-1-3- Results

Figure 1, shows the graph for the average buffer occupancy and figure 2, shows the results for average response time. The orange line represents our simulation result and the blue line refers to the results that are computed by using the equations 1 and 2. In other words, we can say that blue lines refer to theoretical results.

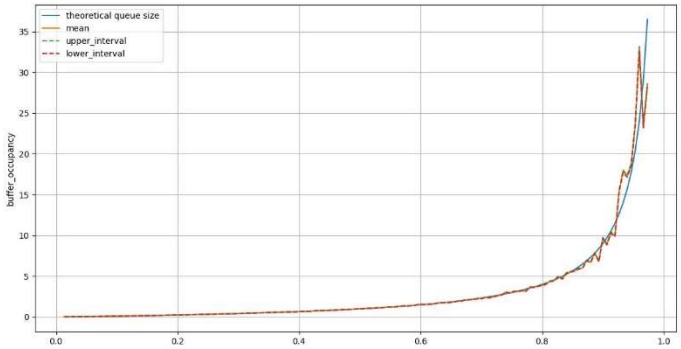

Figure 1: M/M/1 Average Buffer Occupancy

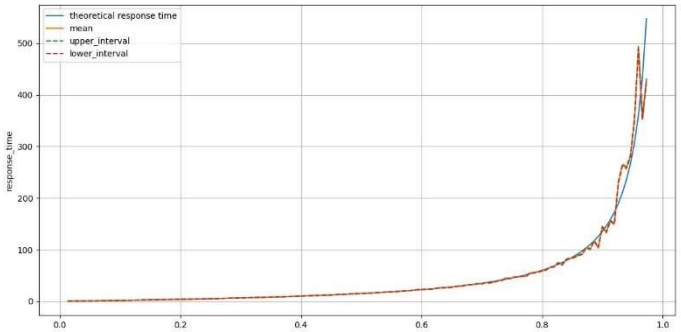

Figure 2: M/M/1 Average Response Time

Finally, the dash lines are related to the lower confidence interval and upper confidence intervals. To calculate the confidence interval, we use confidence level equal to 95 percent. For having a better view on the confidence intervals, we print our results in csv file for both average response time and buffer occupancy at the end of the simulation (Output files can be found in the zip files with the source codes files), by comparing the obtained results and the theoretical results we understand that our simulation is working correctly and it confirms that if ρ approaches to 1 the buffer occupancy and response time goes to infinite.

1-2- The M/M/1/B- Queue

For this part, we simulated the M/M/1/B queue. The difference between this queue and the previous one is the capacity of the queue which is restricted to size B and the system can hold up to B customers. Here, the HTTP requests arrive in batches to server, the size of each batch is uniformly distributed in the interval [a, b]. Like before, arrival rates are exponentially distributed with parameter λ and the service rate with parameter μ.

The lost probability of this system can be calculated from below equation:

$$L = \frac{\lambda \pi_B}{\lambda} = \pi_B = \frac{1-\rho}{1-\rho^{B+1}}\rho^B$$

Equation 3: Loss probability of M/M/1/B Queue

As you can see in the equation 3, when $\rho < 1$ ($\rho = \lambda / \mu$) if B increases to infinite the loss probability will tend to zero. In the case that $\rho > 1$ and B goes to infinite the equation will change to 1- (1/ρ). So, by increasing the ρ we expect to lose more packets.

1-2-1- Inputs

The input parameters are as follows:

```
MIN_RANGE = 1
MAX_RANGE = 15
RANDOM_SEED = 65
INTER_ARRIVAL = 30
SERVICE_TIME = 15
NUM_SERVER = 1
SIM_TIME = 1000
BUFFER_CAPACITY = 45
CONFIDENCE = 0.95
```

1-2-2- Algorithm Description

Most part of the algorithm is like before, except this time we fix the SERVICE_TIME and in every run, we decrease the INTER_ARRIVAL by 0.1. In this way, by every iteration the rate of λ will be higher so we will have higher values for ρ. Please note that unlike previous case, here we need to remove the transient in our simulation. The transition is removed in the beginning of every iteration to reach to data related to steady state faster. This time we also plot the packet loss. Please Note that for x-axis, we use the value of ρ.

1-2-3- Results

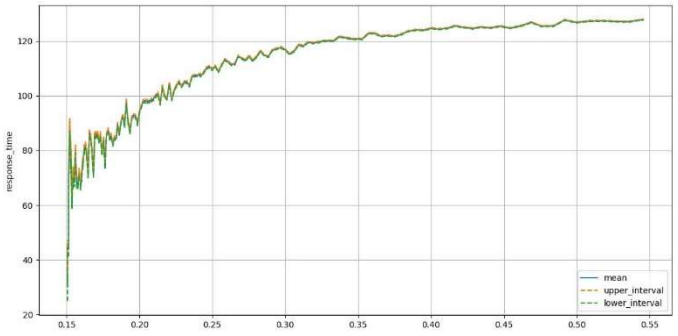

Figure 3: M/M/1/B Average Response Time

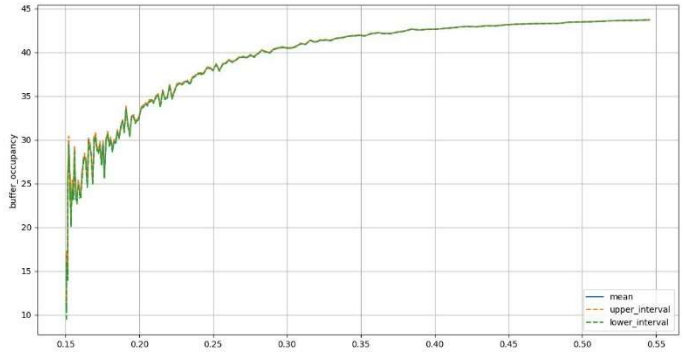

Figure 4: M/M/1/B Average Buffer Occupancy

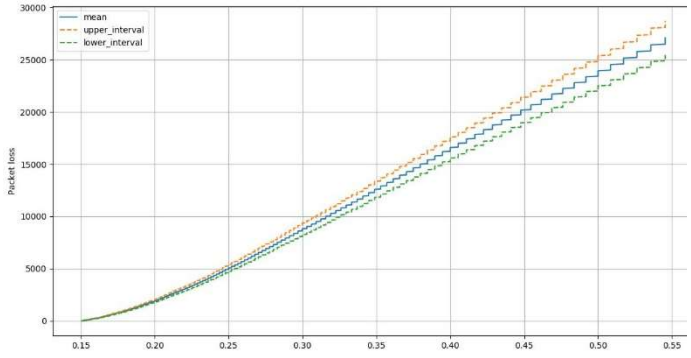

Figure 5: M/M/1/B Packet Loss

Figure 3 and figure 4, show the results for average response time and average buffer occupancy. As we expected by increasing the λ the reaction of the buffer occupancy and response time is the same. The system will reach to its buffer limit faster and it stays in that state, and the response time for the packets in the queue is almost the same so it shows that the system is in the steady state. Finally, figure 5 shows that by increasing the arrival rate and keeping the service time fixed, the average number of packets that are going to be rejected from the queue linearly grows.

Exercise 2: HTTP Accelerator + Web Server

In this section, we have two servers in parallel. The fist server is front- end server that operates as a cash. This server has a high amount of memory to satisfy the popular request and also, a higher rate of service μ1. The HTTP request will arrive to this server in batches, the size of each batch is uniformly distributed in the interval [a, b]. Like before, arrival rates are exponentially distributed with parameter λ. Some requests cannot be satisfied with front- end server, and the packets will be sent to back- end server with probability of p. the service rate in the second server is equal to μ2. Both servers hav a limited buffer capacity B1 and B2 and μ1 >> μ2.

We are going to analyses the response time and buffer occupancy of this system Considering different values for μ1 and μ2 and suggesting a reasonable value for p.

2-1- Inputs

The input parameters are as follows:

```
MIN_RANGE = 1
MAX_RANGE = 15
RANDOM_SEED = 64
INTER_ARRIVAL = 15.5
BUFFER_CAPACITY_1 = 50
SERVICE_TIME = 1.0
BUFFER_CAPACITY_2 = 20
SERVICE_TIME_2 = 5.0
PROBABILITY = 0.30
NUM_SERVER = 1
SIM_TIME = 100000
CONFIDENCE = 0.95
```

2-2- Algorithm Description

Most part of this algorithm is same as the previous section. The value PROBABILITY that we set here, shows the percentage of the packets that are sent to the second server. For example, if we set it to 0.3, 70% of packets will be served with front- end server and only 30% will be sent to

back- end server. We also decrease the INTER_ARRIVAL by 0.1. In this way, by every iteration the rate of λ will get higher so we will have higher value for ρ and at the end for analyzing the p we fix the SERVICE_TIME in both servers. To plot the characteristic of the first server for x-axis, we use the value of ρ, and for the second one and the whole system we used λ.

2-3- Results

We have to analyze the response time related to the whole system. We realized that if we increase the difference between μ1 and μ2, the packets will be served faster in the first server and the second buffer will be utilized faster and the packets will be going to be dropped by the back-end server so this will affect the response time of the system. To avoid packet dropping in the second server we have to set a higher value for the buffer size or decrease the probability value.

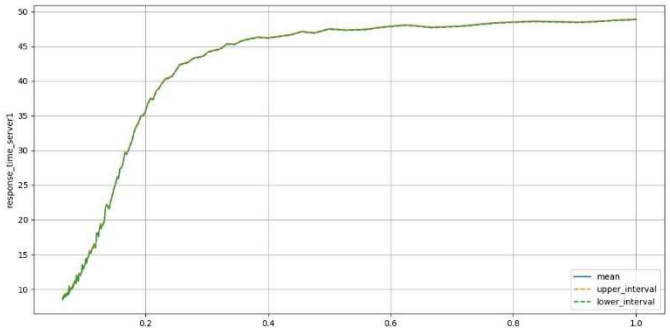

Figure 6: Front- end Response Time

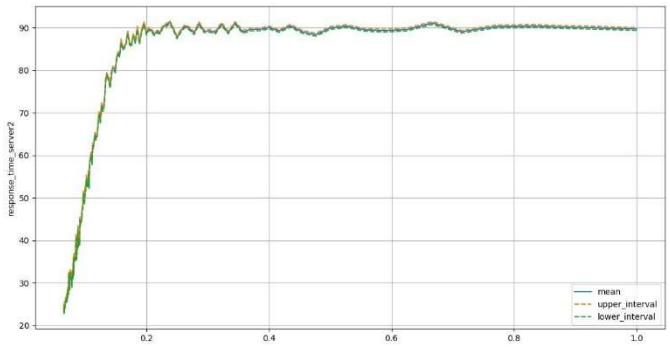

Figure 7: Back- end Response Time

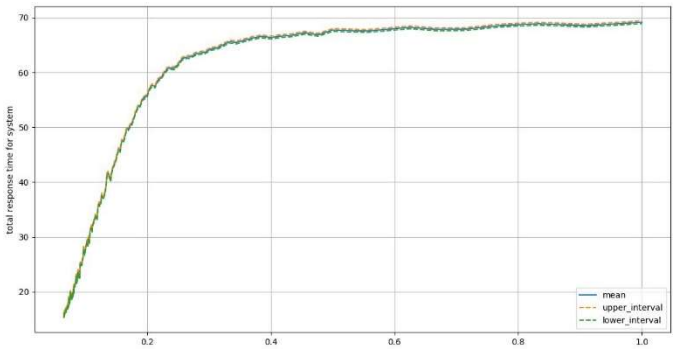

Figure 8: System Response Time

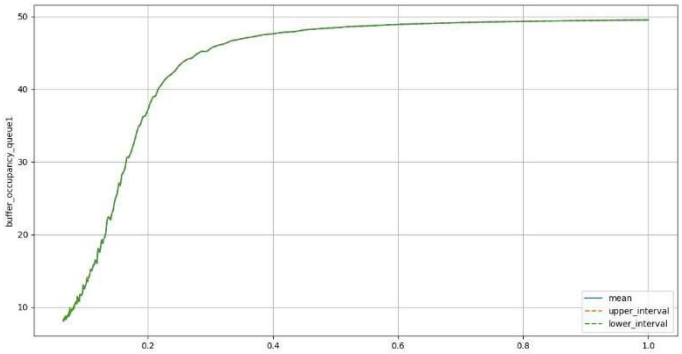

Figure 9: Front- end Buffer Occupancy

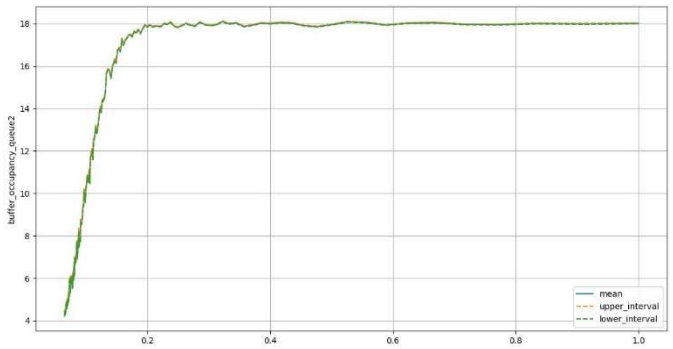

Figure 10: Back- end Buffer Occupancy

As you can see in figures 6,7 and 8 the response time in first server is smaller than the second server and the response time of the whole system is something in the middle. About the buffer utilization, we can say that since we are increasing the arrival rate, the two buffers will become full anyway. Therefore, it is better to avoid this situation that makes the back-end server buffer saturated. Because the packet will

be dropped from the second queue and the request will be sent to the whole system again and this can make the rate of congestion higher in the whole system.

At the end, we can conclude that it is better to reduce the difference between μ1 and μ2 and not to set the value far from each other for these values. Because the second server is slower than the first one it is better to set the buffer size a little bit higher than the other one and for p value we can say that it is better to set it between 0.35 and 0.25. In this way, 65% to 75% of the packets will be served by the front- end server.

LAB 2

Exercise 1: Simulation of Cloud Storage Synchronization

Scenario

In the first part of this lab is we simulate a Cloud Storage System using two possible approaches: the first one follows a server-based architecture, while the second one introduces a p2p synchronization. The main objective is to find the differences between these two systems, in particular focusing on workload.

2-1- Simulation of cloud storage download

This system works similarly to Dropbox: devices share files using shared folders.

2-1-1- The scheme

The network architecture is provided by the given skeleton, only the number of devices has been modified in order to compare the behaviour of the network in different cases.

The task gives a cycle scheme that each devices must follow:

Go online and check if there are new files to download

Wait for an inter-upload time

Upload session:

1- Upload a file (if enough time)

2- Wait for an inter-upload time

Go offline and wait for an inter-session time

Moreover if a device is on-line and a new file is available, the device should download it immediately.

The times are given by given distributions.

2-1-2- The code

The code simulates the scheme and it has the following features:

Upload and download throughputs have the same value, randomly chosen from given files;

Throughput changes in every session;

At the beginning of the simulation all devices are in “inter-session” time and the shared folders are empty;

Devices check the availability of new files also during the inter-upload session;

Devices upload in the same folder during the all session;

IDs and sizes of files are randomly chosen from UbuntuOne values.

The most important part of the algorithm is the “Device” class which make each device an independent process that calls upload and download methods. The first one is called “uploading_session” and it is composed of two part:

Inter-upload: The inter-upload is given by a normal distribution and it is divided into 100 small slots in order to check one time for each slot if new files are available; if it happens that the device starts to download these files, it postpone the upload of others. This checking is inside the “downloading” method that is continuously called.

Uploading: the device randomly chooses a file and thanks to the session throughput, it evaluates the transfer time. If it finishes before the end of session it uploads the file, otherwise the file is lost.

The download method (“downloading”) is called when a device goes on-line and it checks if there are new files and then it starts to download. Also in this case the transfer time is evaluated through the session throughput and if the download is

finished after the end of session the file is not saved and it must be entirely downloaded in the next log-in.

A shared folder notifies the presence of a new file pushing it inside a queue that each device has ("my_queue": since it is not in the queue, devices can download following the inserted order)

The session time and the inter-session time are long distributed variables evaluated in the "connect" method. This one manages the session of the device.

2-1-3- Simulation

We have simulated the system with different number of devices and different number of seeds. In particular we tried with 20, 50, 100 and 250 devices. The simulation time is fixed at 100000.

According to the results, we have to focus on three important values: mean of active devices, traffic entering to the server and traffic exiting from the server. These three values give us the workload on the cloud.

Obviously increasing the number of devices, the traffic on the server increases, but focusing on the different tests with the same number of device we noted that the workload on the server does not depend only on the mean of active devices, but also on the distribution of the shared folders. In fact the number of folders and the way in which they are assigned to devices are not fixed (as said before they follow a negative-binomial distribution): so if many folders are shared by a lot of users, a lot of files needs to be downloaded and so the traffic from the cloud increases, otherwise if devices are connected with a small group of folders, they especially increase the traffic towards the cloud.

Since our device starts with all devices off-line, we need to wait for few times to have a correct value of the mean of connected devices, in particular we measured that time starting from the instant 2000. Removing this transient part we obtain a value around 65% that is larger than the one evaluated starting from the beginning of the simulation.

It is important to point out that file sizes and the throughput values are totally randomly chosen, so they can influence the traffic and in particular these value may cause some peaks in the download/upload traffic.

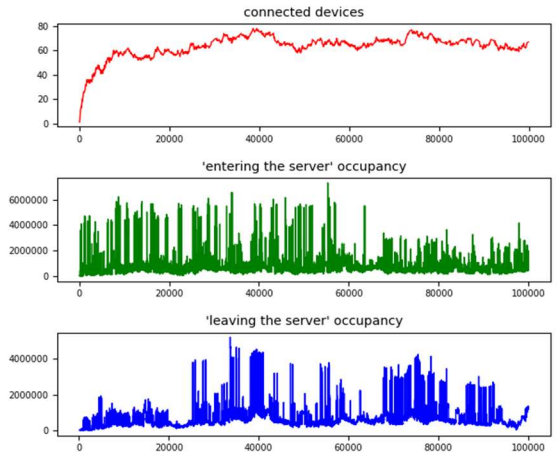

Fig 1: Output with 100 devices (rand.seed=4)

The figure shows the trend of parameters which we consider for a particular case:

The red line represents the connected devices during the simulation: it is possible to see the transient part;

The green graph stands for the workload in terms of upload towards the servers;

The blue one refers to the downloaded data from the server.

It is important to remember that devices cannot download and upload data simultaneously, this feature contribute to define the trend of the server occupancy.

Exercise 2: Synchronization using P2P

This system works like the previous one in terms of upload: users access the cloud and upload files; the important difference is the download part: devices are peers that can share directly files. Using this approach theoretically the server may be used only for "notifying" the presence of new files to be downloaded. The way in which times are simulated is the same (long-normal

distribution) and the generation of the network is given by the skeleton and by the task.

2-2-1- The code

Also in this simulation data of throughput and files (ID and size) are randomly chosen from UbuntuOne data.

Since the system follows the scheme of the previous one, we start talking about the code. It is exactly the same for the upload, but there are some difference concerning the download. It works following these steps (the method is “downloading”):

The device check if there are new files inside its queue;

If a new file is available, the user asks its shared folders the list of devices that have the file and are on-line;

The device connects with the ones inside the list with a different throughput for each connection; the file is divided by the number of devices inside the list;

Thanks to the recursive algorithm (“checking_time”), it is possible to check if each peer stays on-line for all the transfer time; otherwise times are evaluated again because the size changes for each connection. The total transfer time is the largest transfer time. If at the end of checking there are not devices available to complete the transfer or the transfer finishes after the end of the session, the download fails and it must be done entirely at the beginning of the next log-in;

If there are not peers available to transfer the file, the device downloads it from the cloud.

Again “downloading” is called at the beginning of the session and during the inter-upload period in order to check if new files are inside the downloading queue.

Our code is created in order to simulate a system that has the objective to share as much as possible:

Since a file can be uploaded by more devices, it is possible to download a file shared in a folder from a device that is not sharing the folder that sent the notification but another one.

To better understand this point, see the figure 2:

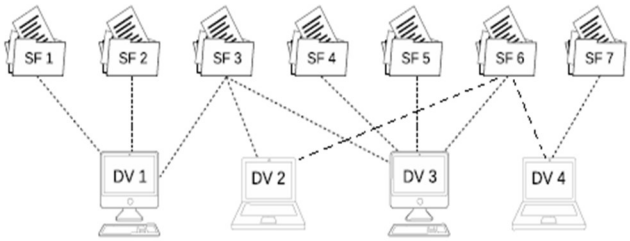

Fig 2

If DV1 shares file A in SF3, DV2 and DV3 have to download it. Suppose that DV4 is already sharing file A with SF7; since DV4 is connected with DV2 and DV3 throws SF6, DV2 and DV3 can download from DV4.

Moreover, we suppose that a peer does not have limitation in the bandwidth but in order to make a more realistic simulation, the values of throughput for each peer connected is divided by the number of devices. Without this precaution there should be the risk of having a very high throughput directed to the device that is downloading because the list that contains the data is the same for the server and the peers. We could choose to divide the session throughput by the number of connected devices, but usually p2p and server-user speeds are different. We considered that a device uses the summation of all throughputs per the largest time transfer.

2-2-2- Simulation

In this simulation we used the same parameters of the previous one (20, 50, 100, 250 devices) in order to compare the results for the same number of device. Simulation time is 100000.

About upload and connected devices we obtained similar results, but we need to focus on the downloading traffic because the code was modified especially on “download” part.

The following figure shows the server workload in terms of devices, download and upload; it is possible to see that the traffic leaving the server is very smaller than the one entering it.

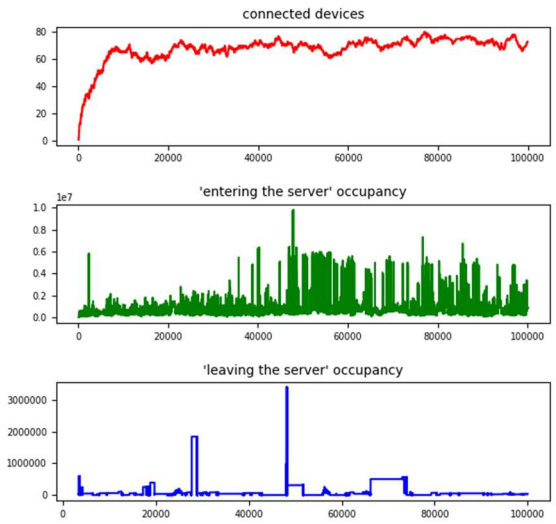

Fig 3: "server traffic",output with 100 devices (rand.seed=4)

So peers download the major part of files from other peers. In particular, checking the results, this traffic depends on the number of on-line devices and it is around 85-90% of the total traffic.

Again we have to note that the number of devices connected is not the only variable that influence the p2p traffic, in fact the structure of the network is very important. In order to understand this concept we can use an example:

SEEDS	%p2p	mean of dev	d. per transfer	d. with file
1	0.8	30.93	1.17	1.3
2	0.91	30.88	2.46	2.97
3	0.89	34.27	1.58	1.73
4	0.87	34.11	1.79	2.03

The above table reports the results of the simulation with 50 devices and different values of seeds. We can see that for seed=1 and seed=2 the mean of connected devices is very similar, but the percentage of p2p download (on the total download) is very different (11%). We can find the motivation if we consider the number of devices per transfer: the simulation with rand.seed=2 creates an ambient with a lot of devices connected each other.

2-2-3- Consideration

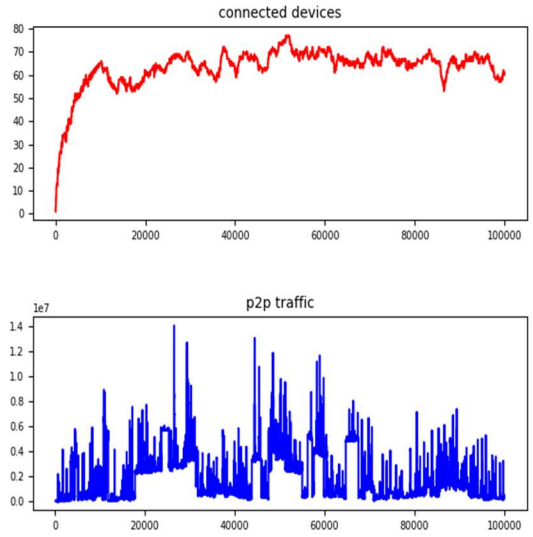

Fig 4: p2p traffic

Comparing the results obtained with two systems, we can say that the p2p system reduce a lot the workload of the server in terms of downloading from it, considering all the obtained values the p2p traffic is about the 90% for obtaining files.

In the above figure on we can see the traffic generated by downloading from peers.

To make this possible it is necessary that peers find on line peers with needed files, in our simulation we found out that the main problem in the p2p synchronization is to find a lot of devices with the needed file, we can see it in the table mentioned in previous part of report. Peers with the file are not many, but there is a little difference from the number of peers that have the file and the ones that are offline during the transfer time.

We can find the mean bandwidth used by peers to download from other peers, if we divide it by the mean number of peers that send data we can obtain the contribution of each devices.